# Alternative Approaches to Data Storing and Processing

Mihkel M. Maran
*Department of Applied Mathematics*
*National Research University "MPEI"*
Moscow, Russia
MaranMM@mpei.ru

Nikolai A. Paniavin
*Department of Applied Mathematics*
*National Research University "MPEI"*
Moscow, Russia
PaniavinNA@mpei.ru

Ilia A. Poliushkin
*Department of Applied Mathematics*
*National Research University "MPEI"*
Moscow, Russia
PoliushkinIA@mpei.ru

***Abstract— Most data processing applications store data using a Database Management System (DBMS). The most widely used DBEs belong to relational. In these engines, data is stored in a number of tables. One of the main relational DBMS interface technologies is Structured Query Language (SQL), but there are alternative approaches, like Query By Example (QBE). Since relational databases have poor performance on some types of data, there have appeared other approaches. They in unite called "No-SQL databases", although these approaches are very different from each other. Graph-based and document-based DBMS belongs to them. In this paper, we describe common implementations of relational and No-SQL DBMS and compare their capabilities using a practical example.***

***Keywords— Databases, Data Bases Management System, Relational Databases, SQL, NoSQL, Graph Databases, Document-Oriented Databases.***

## I. Introduction

Developing a software product in the initial stages, the programmer has the problem of selecting one or more DBMS from existing ones. Which one to choose? Not all of them are equal and have the same capabilities. Our task is to determine which database types apply to which tasks.

Today, various companies offer their solutions that can be conditionally divided into two directions, relational (SQL) and non-relational (NoSQL).

The most common are relational MS SQL Server, Db\2, Oracle, MySQL, non-relational: graph Neo4j, Document - oriented MongoDB and others.

The differences between them are how they designed, what types of data support, how they store information.

Relational databases store structured data that typically represent real-world objects. It can be information about a person, grouped in tables, the format of which set at the design stage of the store.

Non-relational databases are structured differently. For example, document-oriented databases store information in the form of hierarchical data structures. We can talk about objects with an arbitrary set of attributes. The fact that a relational database divides into several interrelated tables can be stored in a non-relational database as an integral entity.

The internal structure of various database management systems affects the features of working with them. For example, non-relational databases are better scalable.

## II. Relational DBMS

The direction of relational databases was born in the 1970s at IBM. Dr E.F. Codd proposed the idea of a new relational data model and specific language at IBM. The research project was named "System R". The SQL language became widespread in the 1980s when relational databases became widespread on large computing machines. SQL cannot be fully assigned to programming languages because it contains only the data access operators stored in the database, and there are no operators that control program progress. SQL statements can be run interactively or embedded in programming languages C#, C++. SQL is based on relational algebra operations and allows you to work with sets (relations whose operands are tables) instead of individual table records. Among the strengths of the language, I would like to highlight strict standardization, independence from DBMS, portability from one computer system to another, declarations. Of the shortcomings, the developers E. Codd and K. Deit highlight the complexity of working with large hierarchical structures. The language that was conceived initially for the end-user became development tools for the programmer due to complexity.

Note that a second language QBE for relational databases was developed in parallel with SQL. Moshe M. Zloof proposed this variant. It was more tailored to users who had almost no experience in database development than SQL. He intended to enter data into blank tables on the screen by filling in forms, so he was more attractive to users with minimal training. QBE contained PRINT, INSERT, UPDATE, DELETE operations similar to SQL - SELECT, INSERT, UPDATE, DELETE. But user access control operation GRANT, object creation operation CREATE queried through the SQL interface. QBE has been widely recognized and is now at the heart of MS Access.

## III. Graph DBMS

In the number of applications, the programmer has to work with data that is easier to represent as a graph [1] than as tables. As examples can be transport routes, semantic networks. In these cases, the use of relational DBMS as storage facilities can be complicated and inefficient. Alternatively, it is possible to use non-relational NoSQL solutions with graph DBMS as one direction, e.g. Neo4j, OrientDB, InfiniteGraph.

Neo4j is a NoSQL open source graph DBMS that was first implemented in Scala and Java languages in 2003 by the American company Neo Technology. Neo4j comes in two revisions, Community and Enterprise. The source stored in the public domain on the GitHub along with the full documentation and description. Currently, Neo4j is the most common graph DBMS.

Neo4j uses its storage format, which was specifically designed to store information as a graphical structure. If you compare with graph modelling on relational DBMS tools, this approach increases system efficiency by performing additional optimization. It is worth noting that processing the graph does not require full loading into RAM. In recent versions of the Neo4j, developers have further optimized for SSD drives and recommend deploying the system on them.

To help in working with the database libraries have been implemented for many programming languages popular nowadays, namely, C++, Java, Python, Go, and there are also its REST-API for building distributed scalable web services. You can extend the software interface in two ways. The first method is to use server plugins that can add new resources to the REST interface. The second method consists of unmanaged extensions, which allow full control over the software interface, and can contain arbitrary code, so they used with caution.

Database elements are a collection of objects associated with one or more relationship types. In Neo4j, the main objects are nodes, relationships between them, parameters, and labels. Each node stores its id as well as a set of parameters (key pairs and their corresponding values). Nodes of the same type can have different properties, which is a significant advantage when analyzing unstructured data. Since the data stored as an oriented graph, each relation has its direction from node to node. And all of them have unique parameters. Labels help you group many similar nodes into separate classes. Tagging is one way to improve node indexing, which affects the speed with which data is searched, and ensures that constraint level simulations followed. In comparison with RDBMS, in which indexing is at a lower level, re-indexing in case of data change is faster.

It is Neo4j on the declarative language of Cypher [2] queries. Any query that changes the data graph is a transaction and will always either run entirely or not run at all. Language tools allow data to be presented both as tables familiar to relational models and rendered as a graph, with all nodes and links signed, and to find out which set of parameters stored in the node. It is enough to select the node we want.

Primary operations of the language: CREATE - creates or updates nodes, links; MATCH can behave as both SELECT and CREATE with more stringent default indexing; RETURN and ORDER_BY borrowed from SQL.

## IV. Document-Oriented DBMS.

Document-oriented DBMS provide more opportunities than relational ones. In such systems, the unit of data presentation is a document that has a set of individual attributes. This document can be presented in JSON or XML format. These systems support searching by individual document fields or field fragments, arrays and sub-documents are allowed, and, here, there is usually no predefined data scheme [3,4]. Some values to avoid infrequent redundancy use can be stored in a separate document and referred to other documents. Unlike relational systems, document databases allow you to query document sets using several attribute constraints, can perform aggregate queries, sort results, and support indexing of document fields.

Documentary systems usually do not support ACID semantics, but they vary considerably among themselves at the level of data consistency, atomic operations, and methods of controlling parallel access to documents. Documentary DBMS has a flexible data model, which in some cases is more convenient than a fixed circuit, and is better combined with object-oriented programming, reducing the interlayer between DBMS and the programming language.

One representative of the document-oriented DBMS is MongoDB. The flexible JSON format allows you to add new attribute fields to individual documents. It makes work much easier. If you have to add new columns in Relational Bases, add *"NULL"* to rows when there are no values, the document may not contain this field in the MongoDB, which means the same.

## V. The Examples of Implementation

Let's build in Neo4j as an example of a semantic network – a part of the Social Network. Limit our model into two sets of nodes. The first set contains Users nodes, and the second set contains music groups. Some users related by a friendship relationship (Are_Friends), and users may have their musical preferences (Likes_Music), and they may be the same for different users. In the Neo4j Cypher query language, this looks like this.

```
CREATE (User1:Person {name: "Nickolay Paniavin", born:1996})
CREATE (User2:Person {name: "Poliushkin Ilia", born:1996})
CREATE (User3:Person {name: "Sergei Sirius", born:1990})
CREATE (MusicGroup1: Musician {name: "Scorpions", founded:1965})
CREATE (MusicGroup2: Musician {name: "Queen", founded:1970})
CREATE (User1)-[:Are_Friends]->(User2)
CREATE (User2)-[:Are_Friends]->(User1)
CREATE (User2)-[:Are_Friends]->(User3)
CREATE (User3)-[:Are_Friends]->(User2)
CREATE (User1)-[:Likes_Music]->(MusicGroup1)
CREATE (User2)-[: Likes_Music]->(MusicGroup2)
CREATE (User3)-[: Likes_Music]->(MusicGroup2)
CREATE (User3)-[: Likes_Music]->(MusicGroup1)
MATCH (n) RETURN (n)
```

The graph of this part of the Social Network is represented at Fig.1.

Let's print all the musical groups Sergei likes.

```
MATCH(person{name : "Sergei Sirius" } ) - [: Likes_Music] -> (ms: Musician)
RETURN person, ms
```

The result contains person node Sergei Sirius, edges Likes_Music and group nodes Scorpions and Queen (Fig.2).

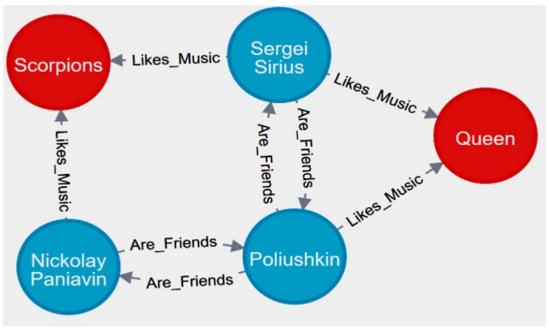


Fig.1. The example of the Social Network.

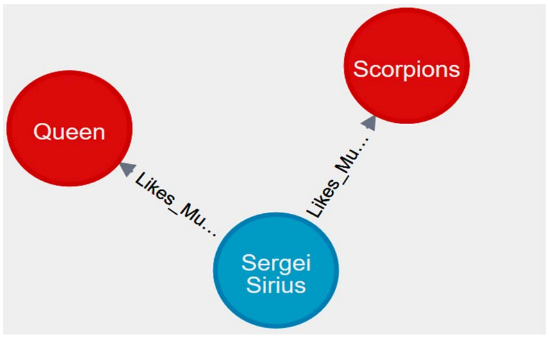


Fig.2. The result of the search as a graph.

And now we bring out all the users who listen to the group Queen and are friends with Sergey.

MATCH (c)-[:Likes_Music]->({name: "Queen"}) , (c)-[:Are_Friends]->({name: "Sergei Sirius"})
RETURN (c)

In this case, only 1 node displayed (Poliushkin Ilia).

It is worth noting that as a result, it is possible to output a separate part of a graph (subgraph), its separate nodes, or a table, as in relational databases.

A graph can store parameters not only in nodes but also on edges (links). In a semantic network, this defined as follows. The musical group that has the most subscriptions (links) has a higher priority, and therefore in the list of recommendations output higher than the rest with lower values of this weight.

On the other hand, using weights, it is possible to regulate the movement of road transport or passenger traffic during morning or evening hours, when it is necessary to distribute it more or less evenly. A route with less time has high priority.

Let's take a look at the shortest path solution in the following example. Note that moving along the edge is only available in one direction [2].

CREATE (a:Loc {name:"A"} )
CREATE (b:Loc {name:"B"} )
CREATE (c:Loc {name:"C"} )
CREATE (d:Loc {name:"D"} )
CREATE (e:Loc {name:"E"} )
CREATE (f:Loc {name:"F"} )
CREATE (a)-[:ROAD {cost:5}]->(b)
CREATE (a)-[:ROAD {cost:5}]->(c)
CREATE (a)-[:ROAD {cost:10}]->(d)
CREATE (b)-[:ROAD {cost:4}]->(d)
CREATE (c)-[:ROAD {cost:4}]->(d)
CREATE (c)-[:ROAD {cost:8}]->(e)
CREATE (d)-[:ROAD {cost:3}]->(e)
CREATE (d)-[:ROAD {cost:8}]->(f)
CREATE (e)-[:ROAD {cost:4}]->(f);

MATCH (n) RETURN (n)

The result is shown at Fig. 3.

The Neo4j in the *"algo"* section delivers several auxiliary algorithms to simplify graph processing. E.g. the Dijkstra function finds the shortest paths from the starting point to the other, but the solution contains only one result. If the shortest path is not the only one, the programmer must control the realization.

MATCH p=(start:Loc {name: 'A'})-[rels:ROAD*]->(end:Loc {name: 'F'})
WITH p, REDUCE(weight=0, rel in rels | weight + rel.cost) as length
RETURN p, length ORDER BY length ASC LIMIT 2

As a result, shown at Fig.4, return all possible the shortest paths in our graph.

It is worth noting that the original graph can be much larger, and the Dijkstra algorithm takes all possible paths. The algorithm incomplete until all of them are the process. It is necessary to take care of the interrupt system by specifying the time_limit.

If you use a different shortestPath algorithm to find the shortest path from the same library, you can specify the length limit, e.g. from 5 to 7.

MATCH path = shortestPath( (start) - [:ROAD] -[5..7]-> (end) ).

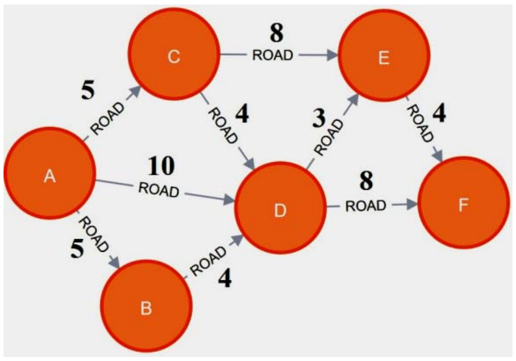
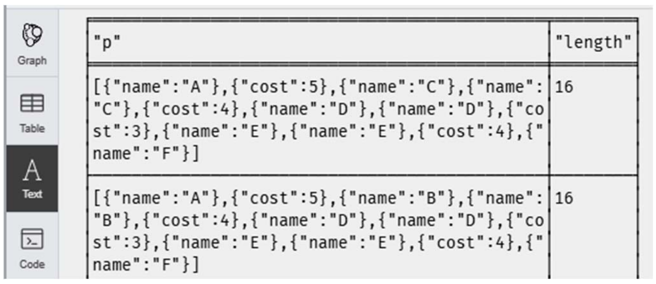


Fig.3. The graph of possible ways between nodes.



| "p" | "length" |
|---|---|
| [{"name":"A"},{"cost":5},{"name":"C"},{"name":"C"},{"cost":4},{"name":"D"},{"name":"D"},{"cost":3},{"name":"E"},{"name":"E"},{"cost":4},{"name":"F"}] | 16 |
| [{"name":"A"},{"cost":5},{"name":"B"},{"name":"B"},{"cost":4},{"name":"D"},{"name":"D"},{"cost":3},{"name":"E"},{"name":"E"},{"cost":4},{"name":"F"}] | 16 |

Fig.4. Two the shortest paths, ACDEF and ABDEF.

Graph in Neo4j also can be processed by means of Java and Gremlin languages.

```
def edge(Graph g, Vertex v1, Vertex v2, int weight) {
e = g.addEdge(null, v1, v2, "connects");
e.setProperty("weight", weight); }
g = new Neo4jGraph("/tmp/dijkstra")
v1 = g.addVertex(1); v2 = g.addVertex(2);
v3 = g.addVertex(3); v4 = g.addVertex(4);
edge(g, v1, v2, 13); edge(g, v1, v4, 20);
edge(g, v2, v3, 3); edge(g, v4, v3, 40);
```

The result is shown at Fig.5.

At once let's find the shortest path between vertexes V1 and V3.

```
import org.neo4j.graphalgo.GraphAlgoFactory;
import org.neo4j.graphalgo.CommonEvaluators;
import org.neo4j.kernel.Traversal;
dijkstra=GraphAlgoFactory.dijkstra(Traversal.expanderForAllTypes(),CommonEvaluators.doubleCostEvaluator("weight"))
path=dijkstra.findSinglePath(((Neo4jVertex)v1).getRawVertex(), ((Neo4jVertex)v3).getRawVertex())
println path
```

The result is as follows:

```
//(1)--[connects,0]-->(2)--[connects,2]-->(3)
weight:16.0
```

And now let's look at the possibilities of MongoDB. Create multiple user documents [4].

```
nick=({"name" : "Nick", "age": 23, "prefMusic" : "Queen" })
db.AreFriends.save(nick)
ilia = ({"name": "Ilia", "age": 23, prefMusic: "Queen", hasFriend: new DBRef('AreFriends', nick._id)})
db. AreFriends.save(ilia)
```

Note, that the first document has no field “prefMusic”.

Let us print all documents in our database with function Find( ) in Command-Line of RoboMongo.

```
db.AreFriends.find()
```

The result is shown at Fig.6.

And now let us print all documents Ilia's friends (Fig.7).

```
db.AreFriends.find({_id: ilia.hasFriend.$id})
```

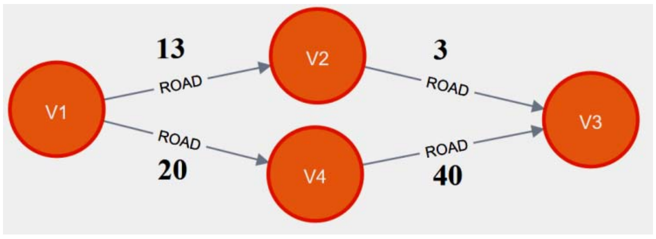
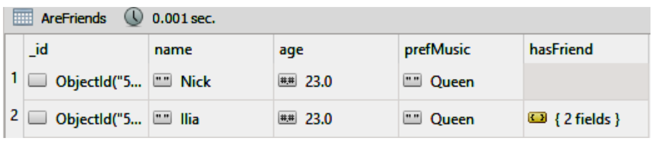


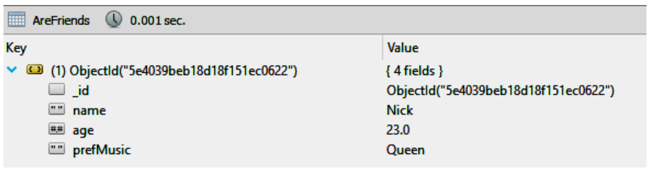
Fig.5. Gremlin generated graph.



AreFriends 0.001 sec.

| | _id | name | age | prefMusic | hasFriend |
|---|---|---|---|---|---|
| 1 | ObjectId("5... | Nick | 23.0 | Queen | |
| 2 | ObjectId("5... | Ilia | 23.0 | Queen | { 2 fields } |

Fig.6. All documents in AreFriends database.



AreFriends 0.001 sec.

| Key | Value |
|---|---|
| (1) ObjectId("5e4039beb18d18f151ec0622") | { 4 fields } |
| _id | ObjectId("5e4039beb18d18f151ec0622") |
| name | Nick |
| age | 23.0 |
| prefMusic | Queen |

Fig.7. The results of the search.

The advantages of MongoDB include a non-strict template of all documents and the ability to implement the data schema directly in the application.

## CONCLUSION

The part of this work, alternative approaches to data processing were considered. The study assumes supplementing relational databases and not insists on their replacement.

All data, which in one way or another presented in the form of graphs, can be entered into the Neo4j and with it is possible to work. You can also fill in any other database, relational or non-relational, with this data. The advantage of a graph DB represented on systems with a large number of nodes and links. It is worth noting to solve the problem bases on social networks, transport maps, and it is more convenient to use graphs that allow obtaining faster and more efficient solutions than relational DBMS. Despite the advantages, the graph database has a significant disadvantage - the amount of disk space consumed. The programmer faces a new task of choosing between a quick solution and the organization of available memory, which is limited.

When data is poorly structured and fits into a database without a fixed schema, Document-Oriented DBMS can take place. MongoDB uses all available memory on the computer as a cache automatically. However, if it needs, MongoDB provides cached memory to the other process. DBMS uses as much free memory as possible, replacing the disk as necessary. Documents can be grouped into collections. Therefore, they can be considered a distant analogue of relational DBMS tables, but collections can contain other collections. Because documents in a collection can be arbitrary, it is best to combine documents with a similar structure into a collection for more efficient indexing. As the query result, it can be documents or their parts on demand.

Thus, choosing a DBMS, it is necessary to proceed base on the initial data and sets of requirements for the solution of the task. The selected DBMS can be used in the training program for programmers in the course “Databases”.

## REFERENCES

[1] Ian Robinson, Emil Eifrem, Jim Webber "Graph Databases. New opportunities for connected data", O`REILLY, 2016, 256p.

[2] Neo4j open user manual, https://neo4j.com/docs/, 17.10.2019.

[3] Karl Seguin "The Little MongoDB book", Attribution-NonCommercial 3.0 Unported, 2016, 66p.

[4] MongoDB manual full version, https://proselyte.net/tutorials/mongodb/ , 17.10.2019 (in Russian).